BPF ASM

https://www.kernel.org/doc/Documentation/networking/filter.txt

Element	Description
A	32 bit wide accumulator
X	32 bit wide X register
M[]	16 x 32 bit wide misc registers aka "scratch memory store", addressable from 0 to 15

Instruction	Addressing mode	Description
ld	1, 2, 3, 4, 12	Load word into A
ldi	4	Load word into A
ldh	1, 2	Load half-word into A
ldb	1, 2	Load byte into A
ldx	3, 4, 5, 12	Load word into X
ldxi	4	Load word into X
ldxb	5	Load byte into X
st	3	Store A into M[]
stx	3	Store X into M[]
jmp	6	Jump to label
ja	6	Jump to label
jeq	7, 8, 9, 10	Jump on A == <x>
jneq	9, 10	Jump on A != <x>
jne	9, 10	Jump on A != <x>
jlt	9, 10	Jump on A < <x>
jle	9, 10	Jump on A <= <x>
jgt	7, 8, 9, 10	Jump on A > <x>
jge	7, 8, 9, 10	Jump on A >= <x>
jset	7, 8, 9, 10	Jump on A & <x>
add	0, 4	A + <x>
sub	0, 4	A - <x>
mul	0, 4	A * <x>
div	0, 4	A / <x>
mod	0, 4	A % <x>
neg		!A
and	0, 4	A & <x>
or	0, 4	A | <x>
xor	0, 4	A ^ <x>
lsh	0, 4	A << <x>
rsh	0, 4	A >> <x>
tax		Copy A into X
txa		Copy X into A
ret	4, 11	Return

Addressing mode	Syntax	Description
0	x/%x	Register X
1	[k]	BHW at byte offset k in the packet
2	[x + k]	BHW at the offset X + k in the packet
3	M[k]	Word at offset k in M[]
4	#k	Literal value stored in k
5	4*([k]&0xf)	Lower nibble * 4 at byte offset k in the packet
6	L	Jump label L
7	#k,Lt,Lf	Jump to Lt if true, otherwise jump to Lf
8	x/%x,Lt,Lf	Jump to Lt if true, otherwise jump to Lf
9	#k,Lt	Jump to Lt if predicate is true
10	x/%x,Lt	Jump to Lt if predicate is true
11	a/%a	Accumulator A
12	extension	BPF extension

User space applications include <linux/filter.h> which contains the following relevant structures:

```
struct sock_filter {    /* Filter block */
        __u16   code;   /* Actual filter code */
        __u8    jt;     /* Jump true */
        __u8    jf;     /* Jump false */
        __u32   k;      /* Generic multiuse field */
};
```

Such a structure is assembled as an array of 4-tuples, that contains a code, jt, jf and k value. jt and jf are jump offsets and k a generic value to be used for a provided code.

```
struct sock_fprog {                     /* Required for SO_ATTACH_FILTER. */
        unsigned short              len; /* Number of filter blocks */
        struct sock_filter __user *filter;
};
```

For socket filtering, a pointer to this structure is being passed to the kernel through setsockopt(2).

Summary of system calls:

 * *setsockopt(sockfd, SOL_SOCKET, SO_ATTACH_FILTER, &val, sizeof(val));*
 * *setsockopt(sockfd, SOL_SOCKET, SO_DETACH_FILTER, &val, sizeof(val));*
 * *setsockopt(sockfd, SOL_SOCKET, SO_LOCK_FILTER,   &val, sizeof(val)).*